Architectural Reference Models in Middleware Solutions

SOA, Web Services and Molecular Messengers
Keeping Your Architectural Perspective

Bill Nadal, CTO – Herzum Software
OMG MDA SOA Web Services Conference
March 23rd, 2005, Orlando, FL
Overview

- Herzum Software Overview
- Overview of Architectural Styles
- COSM™ Levels of Information Exchange
- COSM™ Interoperability Reference Model
- Evolution of Architectural Styles
- Service Based Approaches
- Service Based Platform (Component Execution Environment)
- Future Middleware and SOA Approaches
About Herzum Software

- International consulting group, present in US and Europe (Italy, UK, France, Turkey. Partners in Poland, Sweden, South Africa).
- Premier supplier of services for Enterprise Architecture and Technology Strategy, SOA integration and implementation, and agile software manufacturing
- Specialized in tactical and strategic architectural and organizational migration to new technologies and agile outsourcing
- Unique experience in component-based development and Service Oriented Architectures. Ideal partner for jump-starting and following through strategic developments
- Extensive network of strategic alliances with service and product companies
- Capable of handling projects and organizations from startups to Fortune 100
Herzum Software Offering

A comprehensive set of services including:

- IT Strategy and Enterprise Architecture (small & large enterprises)
- SOA and Enterprise Integration
- Software Architecture and advanced technologies
- Agile Software process definition and rollout
- Extensive curriculum of advanced technology and architectural courses ("Herzum Software: where architects learn to architect")
- Mentoring on advanced component-based and Web services technologies
- Software Development
- Agile Outsourcing
Software architecture comes in different “styles”. At a high level, examples of main styles are: object-oriented architecture, distributed-object architecture, component-based architecture, service-based architecture, service-oriented architecture.

Each style comes with specific architecture characteristics, addresses specific problems, and has limitations addressing other problems.

Typically, a good architecture uses a combination of styles.

Technologies at times imply or suggest, for example through specific examples, a certain way of applying the technology: the implicit *style* of that technology.

- Technologies can be applied very differently from the implicit *style*.
Usage of the *Service* Term

- Technologists and non-technologists both use the term “service”
  - Technologists: an interface that can be called at run-time
  - Non-technologists: an offering provided by a business, e.g. the ability to reserve a flight (a “business service”)

- *Service* will be used in the Technologist sense in this presentation unless otherwise specified
  - Technically speaking, all distributed technologies with an API-like paradigm can be called “service-based”
    - *E.g. RPC, DCE, CORBA, RMI, DCOM are “service-based”*
Example Characteristics of SOAs

- Standard-based (XML related standards – strongly tagged)
- (Continued) Emphasis on data and information exchanges
- Emphasis on registries and mediation architectures
- Shifting from interfaces to contracts
- Technical and (...future) business negotiation
- Ontologies and other semantic aspects
- From “interface” to “interoperability architecture problem”
- New security challenges
- Allow (and require) new management solutions
- Mindset switch from “install and use” to “find and invoke”
- Very loose coupling
Early SOA and Post SOA Models :>)

Early SOA
First Diagram of the Internet
UCLA to CERN (ARPA)

Post SOA
Genetic Dispatching w/Protein Messengers

Source: http://www.stigmergicsystems.com/F

THE ARPA NETWORK
SEPT 1969
1 NODE
Different levels of information exchanges typically require:
- different architectural approaches
- different technologies
- different standards

Information exchanges
- Intra-application (L1)
- Inter-applications, intra-enterprise (L2)
- Inter-enterprises (between partners) (L3)
- Inter-communities (L4)
Typical Coupling Characteristics

- **Intra-Application (L1):** Tightly coupled applications

- **Inter-Applications, Intra-Enterprise (L2):** Tightly coupled applications or loosely coupled applications (for example through messaging)

- **Inter-Enterprises, Intra-Community (L3):** Loosely or Very Loosely coupled applications
  - Web services

- **Inter-Communities (Federated Enterprise Systems/Applications – L4):** Very loosely coupled systems/applications
  - Web services
Examples: L2-L3 Architectural Choices

L2 : Inter-applications, intra-enterprise
- Quite coarse grained exchanges (can be less than document)
- Sub-transactional or transactional information exchanges

L3 : Inter-enterprises
- Coarse-grained documents
- Transactional boundaries managed by individual enterprises
- Business transactions managed simply by compensation actions
## Evolution: Computing Systems and Applications

<table>
<thead>
<tr>
<th>Time Period</th>
<th>System Type</th>
<th>Usages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-1965</td>
<td>Decentralized Batch Systems</td>
<td>Tabulating, 1401 on EVERY FLOOR, Information Systems</td>
</tr>
<tr>
<td>1965-1975</td>
<td>Centralized Batch Systems</td>
<td>Transistors, Mainframes, DUMB TERMINALS</td>
</tr>
<tr>
<td>1975-Present</td>
<td>Distributed Access Computing</td>
<td>PCs &amp; LANS, ONLINE TRANSACTION PROCESSING &amp; TIMESHARING</td>
</tr>
<tr>
<td>1982-Present</td>
<td>End User Computing</td>
<td>PERSONAL PRODUCTIVITY &amp; GROUPWARE, Intra-Enterprise Systems</td>
</tr>
<tr>
<td>1990-Present</td>
<td>Client/Server Computing</td>
<td>Enterprise Networks, e-BUSINESS &amp; COLLABORATIVE COMPUTING</td>
</tr>
<tr>
<td>1996 Forward</td>
<td>Web-Based Computing</td>
<td>Internet &amp; the Web, WIRELESS COMPUTING</td>
</tr>
<tr>
<td>2001 Forward</td>
<td>Ubiquitous Computing</td>
<td>End user-driven e-business, Security &amp; Privacy, Federations of Enterprises</td>
</tr>
</tbody>
</table>

Adapted from: Peter Fingar
COSM™ Interoperability Reference Model

Development Life Cycle Aspects

Functional Reference Model

Semantics

Functional Interfaces

Structural Infrastructure

Technical Infrastructure

Technical Interfaces

(For more info, see free download at www.herzumsoftware.com or “Business Component Factory” book)
Application-to-application within an Enterprise

Using same development environment

Implicit

Agreed by phone

Defined by internal functional and component architects

Internally defined, tightly coupled architectural standards

J2EE, .Net, Corba services and facilities

RMI, Corba, .NET Remoting, (XML?)

Development Lifecycle Interfaces

Functional Reference Model

Semantic

Functional Interfaces

Structural Infrastructure

Technical Infrastructure

Technical Interfaces
Enterprise-to-Enterprise Development Lifecycle Interfaces

- Technical Interfaces
  - Technical Infrastructure
- Structural Infrastructure
  - ebXML
  - Standard defined, loosely coupled architectural standards
  - Ontology
  - Rarely exist, but should be standard
- Functional Reference Model
  - Functional Interfaces
  - Semantic
- Development Lifecycle Interfaces
  - XML, HTTP, SOAP
  - UML, MOF, XMI, …
Gaps in Web Services Standards Coverage

OMG MDA, UML, XMI, EDOC ...  
(some non-Web Services standards, such as STEP)

ebXML, RosettaNet, ...

ebXML, RosettaNet, (XML Schemas)...

BPEL, SAML, DAML, BTP  
(other OASIS standards)

XML, SOAP, WSDL, UDDI

Dev. Aspects
Func. Ref. Model
Semantic
Func. Interfaces
Struct. Infra.
Tech. Infra.
Tech. Interfaces

Coverage
Architectural Styles

- 1970: Structured Programming
- 1980: Object-Oriented
- 1990: Distributed Objects
- 2000: Enterprise Components
- 2010: Service-Based

- Complexity
- Importance of Architecture
**Brief History of Time: Components**

1970

**Structured Programming**
Cobol, Pascal, Ada, One Technology, Structured Methodologies

1980

**Object-Oriented**
C++, Eiffel, One Technology, OO Methodologies, Very Fine Granularity, Tight Coupling

1990

**Distributed Objects**
CORBA 1, Java, Only Tech. Interop., UML, Fine Granularity, Tight Coupling. Low complexity (no real transactions, security)

2000

**Enterprise Components**
J2EE, CORBA 3, .Net, Coarse Granularity Possible, XML, Loose Coupling, MDA
Brief History of Time: Web Services

1970

Structured Programming
Cobol, Pascal, Ada,
One Technology,
Structured Methodologies

1980

Service-Based
TPM (Tuxedo,
ACMS, CICS, …),
Messaging, n
technologies, System
Granularity, Loose
Coupling

1990

2000

Service-Oriented
Web Services,
Mediation
Architectures,
BTP, System
Granularity, Very
Loose Coupling

2010

Internet, EAI,
eBusiness, …
## Components v/s Services

<table>
<thead>
<tr>
<th>Architectural Perspective</th>
<th>Components</th>
<th>Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>The internal asset of a system (not necessarily shown outside, externals can be Web services or not)</td>
<td>What is seen externally to a system (internals can be components or not)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deployment Model</th>
<th>Components</th>
<th>Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software is physically deployed. “Install and use”</td>
<td>Software “exists” somewhere. “Connect and use”</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Levels of Information Exchange</th>
<th>Components</th>
<th>Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mostly within enterprise</td>
<td>Mostly across enterprises</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Coupling</th>
<th>Components</th>
<th>Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fairly loose. Based on internal standards</td>
<td>Very loose. Based on industry standards</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Communication</th>
<th>Components</th>
<th>Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise-based Protocols (like RMI or .NET remoting)</td>
<td>Internet-based Protocols (like XML over SOAP)</td>
<td></td>
</tr>
</tbody>
</table>
Main Characteristics

- Focus on providing distributed interfaces to existing systems, and connecting them to each other
  - Often through bus-based or hub-and-spoke technology infrastructure
  - Often through simple (technical) mapping to a given distributed technology
- Granularity of “service implementation” is often large monolithic systems
- Individual services are usually transactional (TPM meaning)
- Typically no attempt at providing “autonomous” implementations of services, or reducing dependencies, or associating islands of data
- No layering
Nature of federation in service-based

Each system
- As black-box (run-time, development-time)
- Heterogeneous technologies. Each system manages its own transactions
Service Based Platform
The Component Execution Environment

◆ A factory setup element providing an architectural framework and supporting tools
◆ An architectural framework is a technology independent platform model defined to
  ■ Provide a unique Service Oriented and Component Based Architectural Style across the system
  ■ Address the critical concerns for constructing enterprise class systems
  ■ Integrate best of breed ingredient technologies to meet the changing market needs
◆ The supporting tools enable functional development teams to
  ■ jump start component development
  ■ focus on domain capabilities rather than infrastructure development
  ■ maintain alignment with an overall architectural blueprint
Platform & Functional Web Services

- **Orchestrated Web Services**
  - Providing actual “business-level” (complex) Web Services: getAllCustomerAccounts, reserveTravel

- **Element Web Services**
  - getCustomerAddress, getCustomerAccount, …

- **Utility Web Services**
  - Authorization & authentication, unit of measure conversion, number generation (inside an enterprise or a system)

- **Web Services Technical Infrastructure High level**
  - Distributed Registry Service, Security Infrastructure, Ontology Services, Access Rights Manager, Service Modeler, Business Process & Workflow Modeler, QoS
  - (“Infrastructure Web Services”)

- **Web Services Technical Infrastructure Low level**
  - Simple Lookup & Register
  - Web Services-Specific Network layer
COSM™ Platform Elements

The platform is composed of:

◆ **CEE Kernel.** Responsible for:
  - Component Lifecycle Management
  - Service Discovery
  - Load Balancing
  - Fault Tolerance
  - Providing POJO development environment
  - Security
  - Component and CEE administration

◆ **CEE Facilities.** Responsible for:
  - Exposing simple, technology agnostic interfaces abstracting the underlying technologies
  - Removing the need for the functional developers to understand technology specific nuances

◆ **Utility and Auxiliary Components.** Responsible for:
  - Common base functionality required by enterprise class applications. Examples:
    - *Printing*
    - *Reporting*
    - *Calendaring*
    - *Address Book*
    - *Security*
# Sample COSM CEE™ Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Feature Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logging</td>
<td>Enables distributed collection of log messages generated by the CEE, the platform components, the applications</td>
</tr>
<tr>
<td>Directory</td>
<td>Services and Components are registered and deregistered through the directory – supports autodiscovery and service metadata</td>
</tr>
<tr>
<td>Mega Data</td>
<td>Enable the reliable and progressive delivery of large amounts of data</td>
</tr>
<tr>
<td>Shared Data</td>
<td>Acts as a P2P shared memory space for the distributed CEE and supports replication of state for pervasive services like session management</td>
</tr>
<tr>
<td>Configuration Management</td>
<td>Allows for all components of the CEE to access system and component level centrally managed configuration data</td>
</tr>
<tr>
<td>Smart Proxy / Adapter</td>
<td>The smart proxy framework works behind the scenes, providing enhancements to each CEE service like auditing, tracing, etc.</td>
</tr>
<tr>
<td>Async Invocation</td>
<td>Provides the option of invoking service calls asynchronously or fire-and-forget</td>
</tr>
<tr>
<td>Load Balancing</td>
<td>Monitors instances of the CEE for critical resource utilization. When querying for services, the CEE instance with the most available resources is used.</td>
</tr>
<tr>
<td>Failover</td>
<td>The failover mechanism allows for session-level and invocation-level errors to be trapped and handled</td>
</tr>
<tr>
<td>UUID</td>
<td>Returns a globally unique key (URIs) used to identify object instances within a running cluster of CEE(s)</td>
</tr>
<tr>
<td>XML Processing</td>
<td>Handles intelligent Java to XML marshalling and un-marshalling</td>
</tr>
<tr>
<td>Studio</td>
<td>A plug-in to the Eclipse IDE to allow for component specification and the automation of some CEE development tasks (i.e., asynchronous interface generation.)</td>
</tr>
<tr>
<td>Session Management</td>
<td>Support the creation, distribution, and access to session-level data (e.g. user profiles)</td>
</tr>
<tr>
<td>Security</td>
<td>Provides for the centralized definition and distributed enforcement of a system defined security policies</td>
</tr>
<tr>
<td>Transaction</td>
<td>Support implicit and explicit transaction management</td>
</tr>
<tr>
<td>Persistence</td>
<td>Provides means for the generation and runtime management of object-relational mappings</td>
</tr>
<tr>
<td>Internationalization</td>
<td>Enables the central specification of I18N resource bundles and distributed runtime access on the basis of deployment configuration</td>
</tr>
</tbody>
</table>
Example: “Ingredient Technologies”
**COSM™ Deployment Architecture**

**Admin Console**

- Control Functions
  - Deploy New Components
  - Start/Stop Components
  - View Running Components
  - View Log Files

**COSM Application Server**

- Layered Tiers
  - COSM Components
    - Product
    - Supplier
    - Price
  - COSM Platform
Technology perspective: The COSM™ platform is pre-selecting and integrating the ingredient technologies from the fragmented Open Source product community and providing an application development factory and large-scale runtime environment using a mature component based development approach.

Open Source

- Liferay (Portal)
- Lenya (Content Management)
- CAS – (Security)
- Joram (JMS - async messaging)
- Log4j (logging)
- Quartz (scheduling)
- Jython (scripting)
- Hibernate (ORM)
- Struts (UI)
- JBoss (EJB Server)
- Ant + Cruise Control (build env)
- CVS (change management)
- Atlassian Jira (issue management)
- Sleepycat: Efficient caching
- Drools (Rule Engine )
- BSF (Bean Script Framework)
- JUnit (Unit testing framework)
- Others
CEE Addresses the Application XFRs Contract

- CEE provides the technical infrastructure required to meet the COSM extra functional requirements for:
  - Scalability
  - Fault Tolerance
  - Performance
- CEE accomplishes these goals via an integrated strategy that blends:
  - Federated Clustering
  - Fine Grained Resource Load Monitoring
  - Adaptive (Bind + Invoke) Load Balancing
  - Service Level Heart Beat Monitoring
  - Autonomous Provider Selection
  - SLA based Service Failover
  - State Replication
What is a protein-encoding sequence?

A portion of the DNA provides a "coded message" that describes what pieces ought to be used for making a protein. After the raw message is sent, it is decoded by the protein-construction machinery. The decoding is done using the Genetic Code.

“Key Relay Protein Shapes Cancer Message
Refolding of Messenger Molecules Needed for Breast Tumor Development
A tiny enzyme able to latch onto proteins and change their shape plays a critical and surprising role in promoting some of the most belligerent forms of breast cancer…Among the fastest growing are those that produce an overabundance(1) of the receptor Her2/Neu. Once activated, Her2/Neu sends a signal, via a series of molecular messengers, to an agent lying in wait inside the nucleus that tells the cell to enter the cell cycle. Though researchers have identified some of the middlemen, it is not clear how the message is delivered to the cell cycle gatekeeper, cyclin D1” Source: [http://focus.hms.harvard.edu/2004/July16_2004/oncology.html](http://focus.hms.harvard.edu/2004/July16_2004/oncology.html)  (1) Molecular spam?
Conclusions

- Architectural Reference Models provide key abstractions for positioning middleware solutions
- Information Exchange Models provide context for understanding tradeoffs in levels of coupling and binding models
- The Interoperability Reference Model describes the key architectural layers involved in integrating a service based middleware solution
- The evolution of Architectural Styles into a service based approach has the benefits of a federated architectural model, but with corresponding architectural complexity
- Commercial component and service based platforms (e.g. COSM™ CEE) are emerging mature for certain technology aspects that provide important platform services over various infrastructures
- Reference models provides maps to guide us in understanding future middleware and SOA approaches
Questions?

Herzum Software
175 N. Franklin St., Suite 301
Chicago, IL, 60606  USA


information@herzumsoftware.com

wnadal@herzumsoftware.com

www.herzumsoftware.com